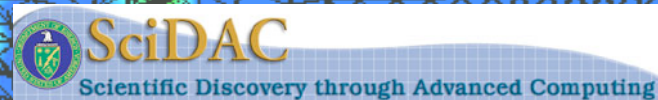


Interoperable Tools for Advanced Petascale Simulations (ITAPS)

Tutorial Overview Presentation

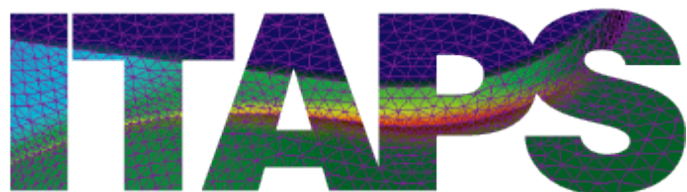
August 2010



BROOKHAVEN
NATIONAL LABORATORY

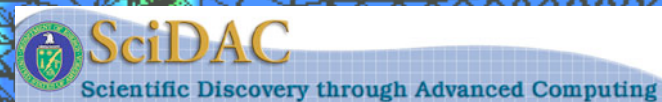


LLNL-PRES-407129



INTEROPERABLE TOOLS FOR ADVANCED PETASCALE SIMULATIONS

Introduction



Rensselaer



THE UNIVERSITY OF
BRITISH
COLUMBIA

BROOKHAVEN
NATIONAL LABORATORY

OAK RIDGE NATIONAL LABORATORY

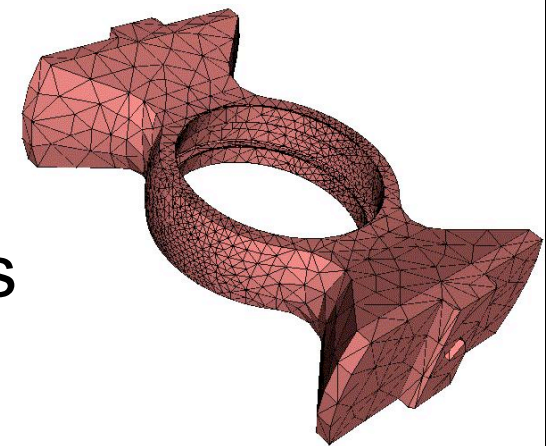
Pacific Northwest National Laboratory

Unstructured meshes

- An unstructured mesh
 - A piece-wise space/time domain decomposition over which the simulation is to be run
 - General topology-based mesh representation consists of
 - 0-3 D topological entities (vertices, edges, faces and regions)
 - Connectivity between entities called *adjacencies*
 - Mesh data structure provide services to create and/or use the mesh data



Geometric domain



Mesh

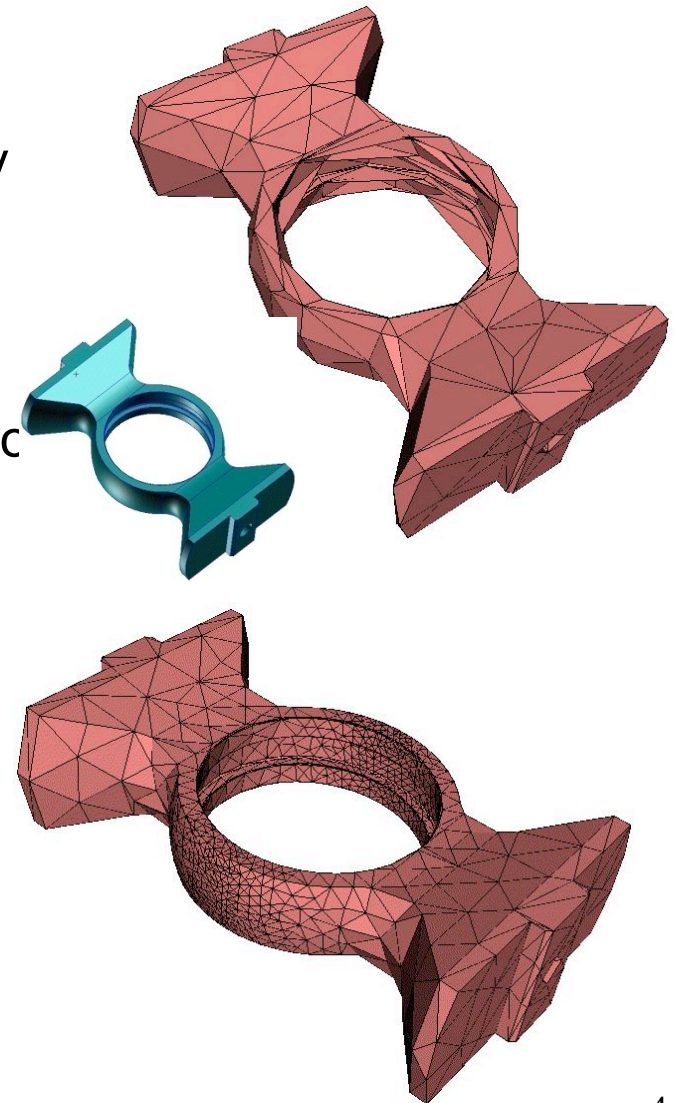
Unstructured mesh methods are commonly used for numerical simulation

- **Some Advantages**

- Meshes of mixed topologies and order easy
- Mesh adaptation can account for curved domains
- General mesh anisotropy can be obtained
- Easy to create strong mesh gradations with special numerical techniques
- Alignment with multiple curved geometric features

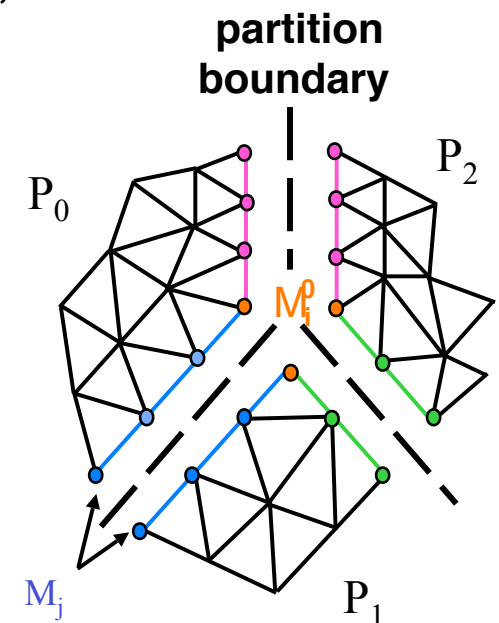
- **Some Disadvantages**

- Data structures larger and more complex
- Solution algorithms can be more complex



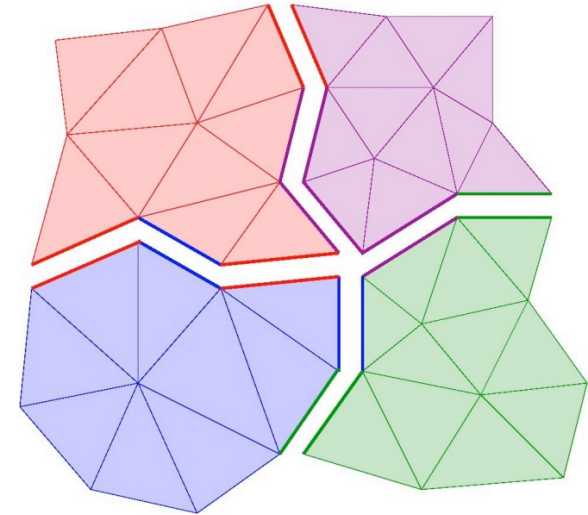
Unstructured Meshes on Parallel Computers

- Typically the mesh is distributed over independent memories
- A mesh partition groups mesh entities and places them into parts
- Applications using a partitioned mesh need
 - Communication links for between “shared” mesh entities on neighboring parts
 - Ability to move mesh entities between parts (while maintaining links)
 - Algorithms to maintain load balance of parts which minimizing communications



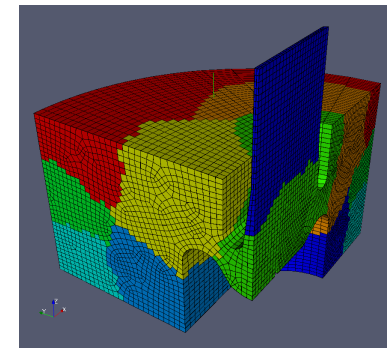
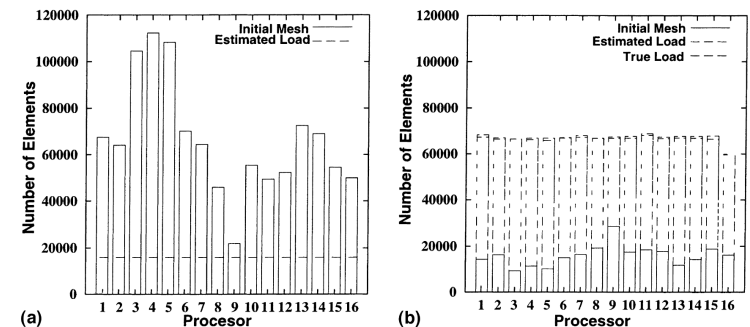
Distributed mesh representations have several functional requirements

- Entity ownership
 - Each mesh entity is owned by exactly one part
 - Ownership imbues right to modify
 - Ownership is not static during the course of a simulation
 - Repartitioning
 - Local micro-migration
 - Some entities have read-only **copies** on other parts (e.g. along part boundaries and ghosts)
- Communication links
 - Efficient mechanisms to update mesh partitioning and keep the links between parts are mandatory



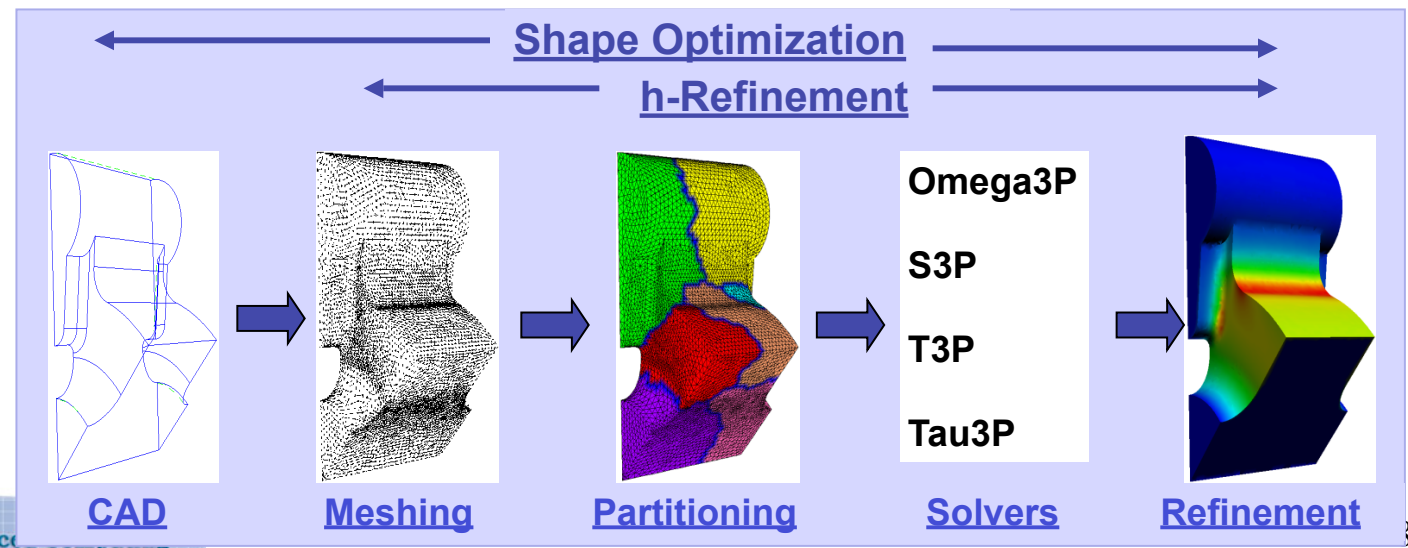
Key issues that must be addressed for parallel computation

- Scalability
 - Load balance as the mesh changes
 - Low communication overhead costs
 - Efficient use of distributed memory
- Function
 - Consistent, correct mesh operations
 - Management of complex communication schedules
- Performance
 - Near optimal serial efficiency on each processor
 - Minimal overhead when using general tools relative to native implementations



Basic parallel solution on unstructured meshes has several key steps

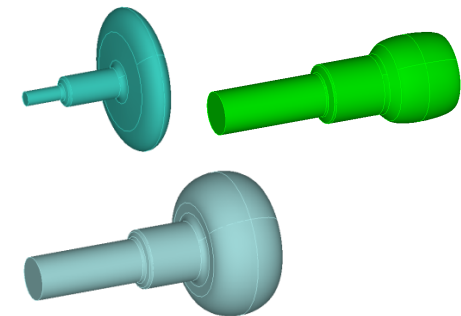
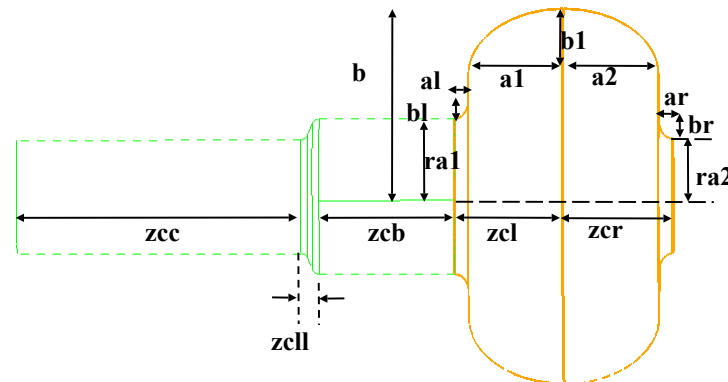
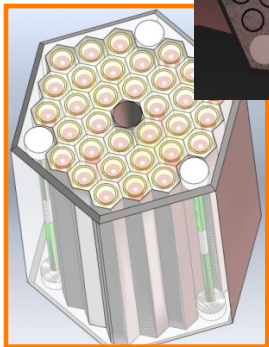
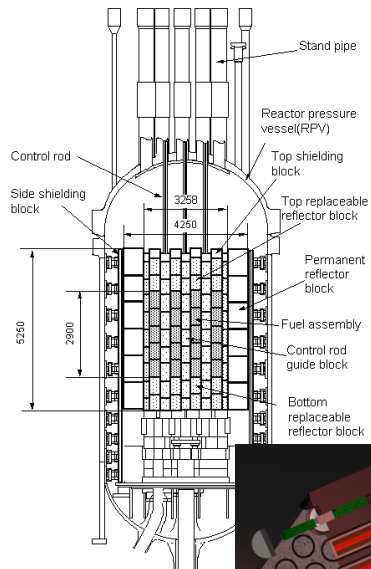
Construct the initial mesh (serial or parallel)
Improve the mesh using **smoothing** and **swapping**
If necessary, **(re)partition** the mesh across processors
Solve the PDE on mesh and estimate the error
While error > tolerance
 Refine, coarsen, improve and repartition the mesh
 Solve the PDE on the mesh and estimate the error
End



Parallel solution is further complicated by the needs of advanced simulations

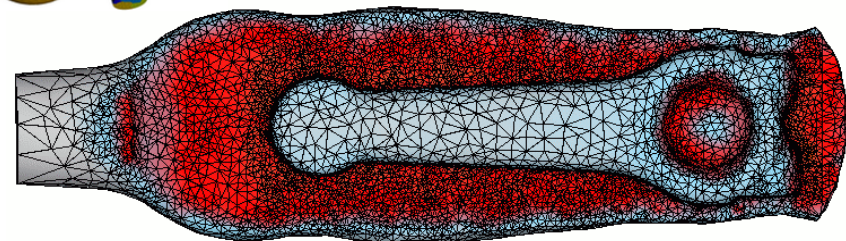
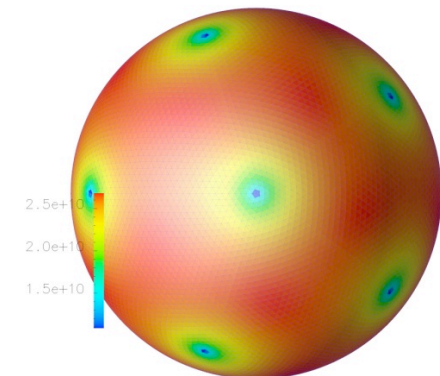
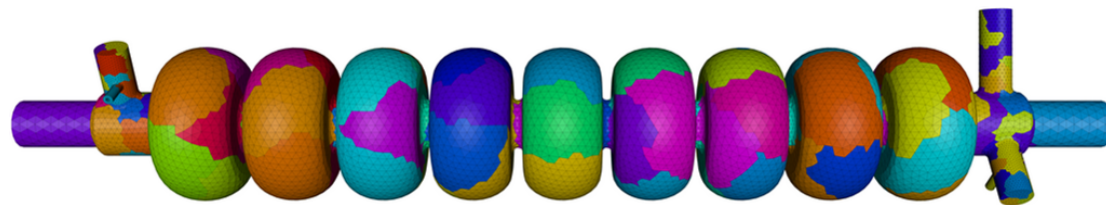
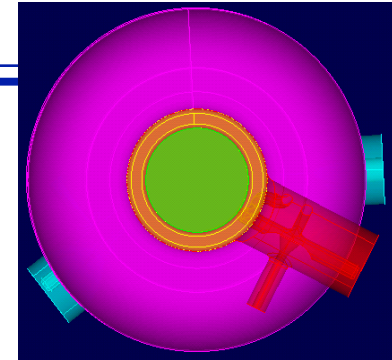
Examples:

- Design optimization requires geometry modification, remeshing, derivative computations
- Multi-physics applications require mesh to mesh transfer, interpolation methods, sophisticated adaptive methods



The ITAPS team has developed tools to address these needs

- CAD interaction: CGM
- Mesh generation: GRUMMP, NWGrid
- Mesh databases: FMDB, MOAB
- Mesh improvement: Mesquite, swapping tools
- Parallel Adaptive loops: FMDB, NWGrid, MeshAdapt
- Front tracking: Frontier
- Partitioning: Zoltan



While these tools exist, significant challenges remain

Developing and using these technologies requires significant software expertise from application scientists

- Difficult to improve existing codes
- Difficult to design and implement new codes

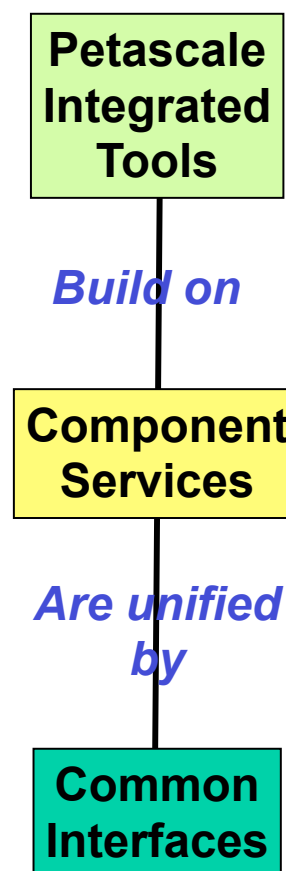
These tools all meet particular needs, but

- They do not interoperate to form high level services
- They cannot be easily interchanged in an application

The ITAPS center is developing key technologies to ease the use of advanced meshing tools on large-scale parallel computers

ITAPS uses a component-based approach to address these challenges

- Develop and deploy key mesh, geometry and field manipulation *component services* needed for petascale computing applications
- Develop advanced functionality *integrated services* to support SciDAC application needs
 - Combine component services together
 - Unify tools with *common interfaces* and *data model* to enable interoperability
 - Interfaces are implemented on top of existing mesh databases
- Work with key application teams to insert ITAPS technologies into simulations



ITAPS produces mesh services that meet application needs

**Petascale
Integrated
Tools**

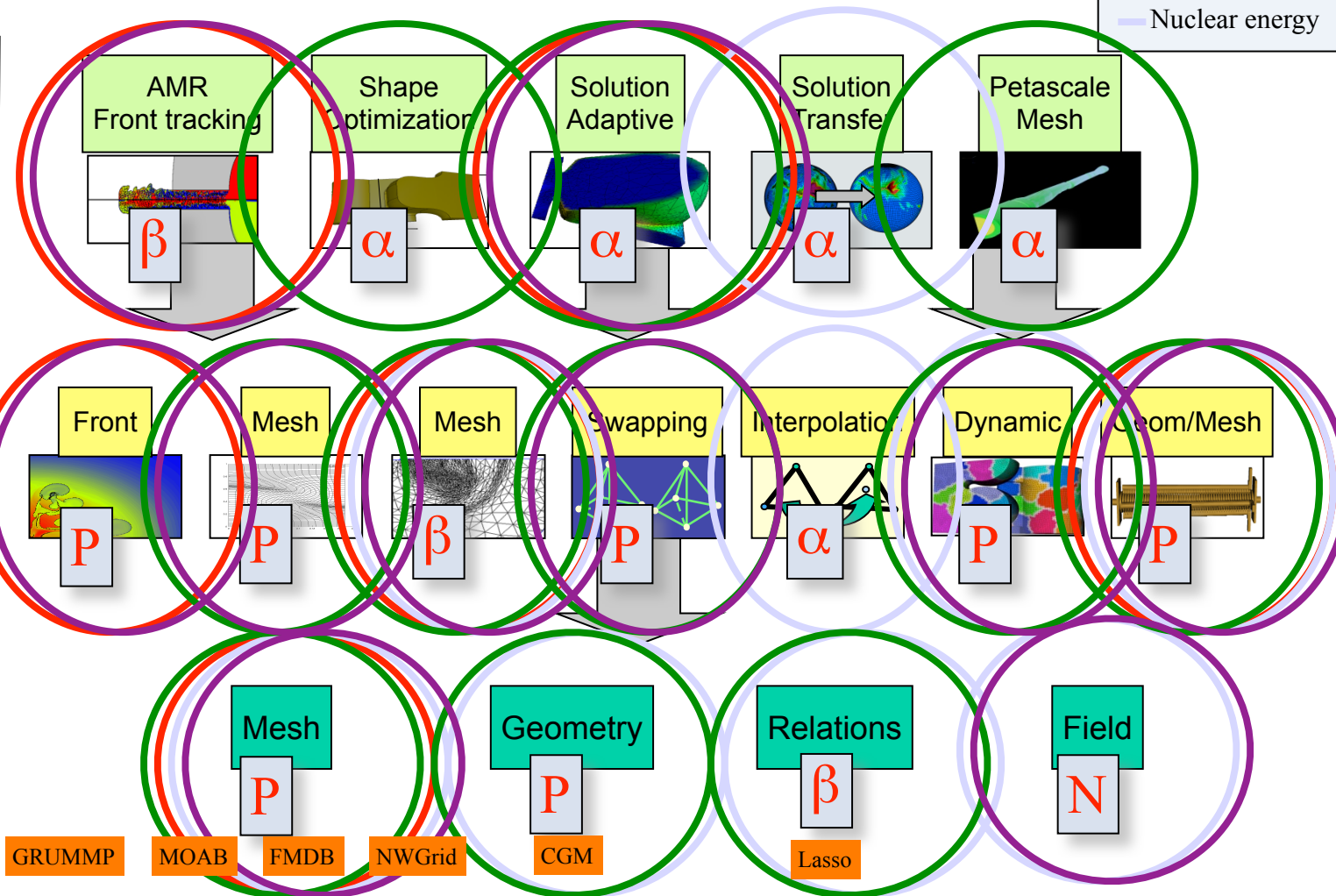
Build on

**Component
Tools**

Are unified by

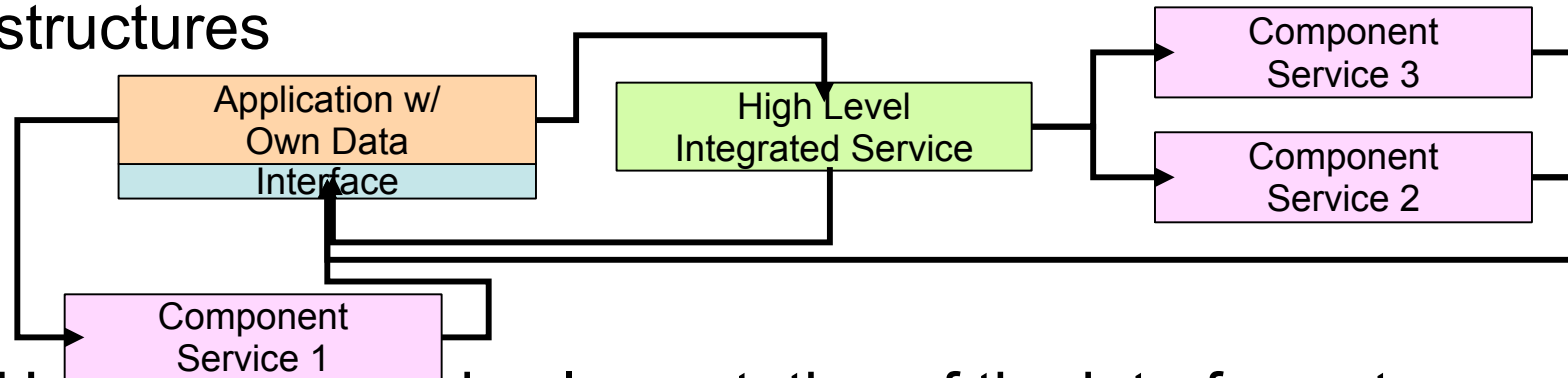
**Common
Interfaces**

to...

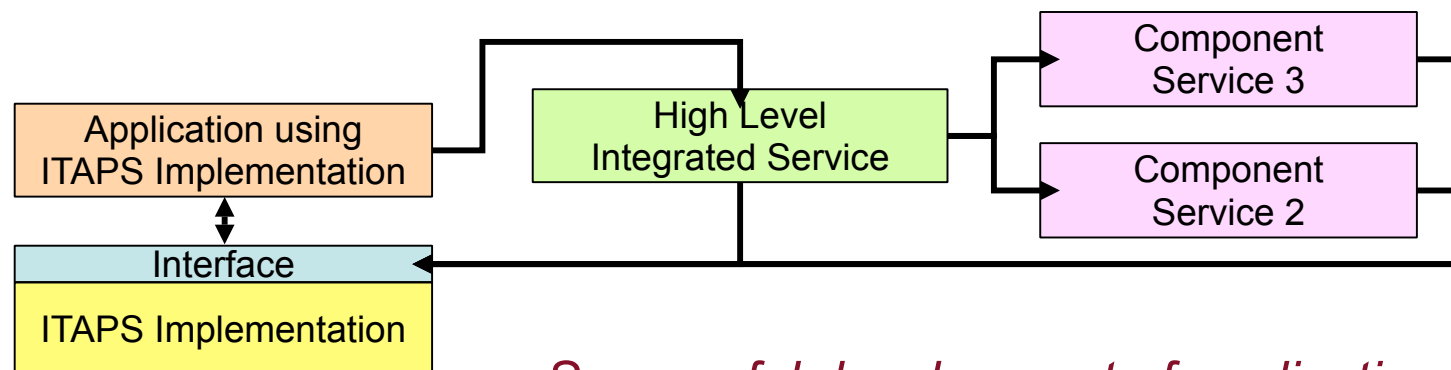


Applications can access ITAPS services in two ways

- Implement ITAPS interfaces on top of application data structures

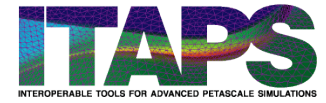


- Use a reference implementation of the interfaces to provide access to ITAPS services at the cost of a data copy



Successful development of applications has been accelerated by close collaboration.

There are several advantages of the component-based approach ITAPS uses



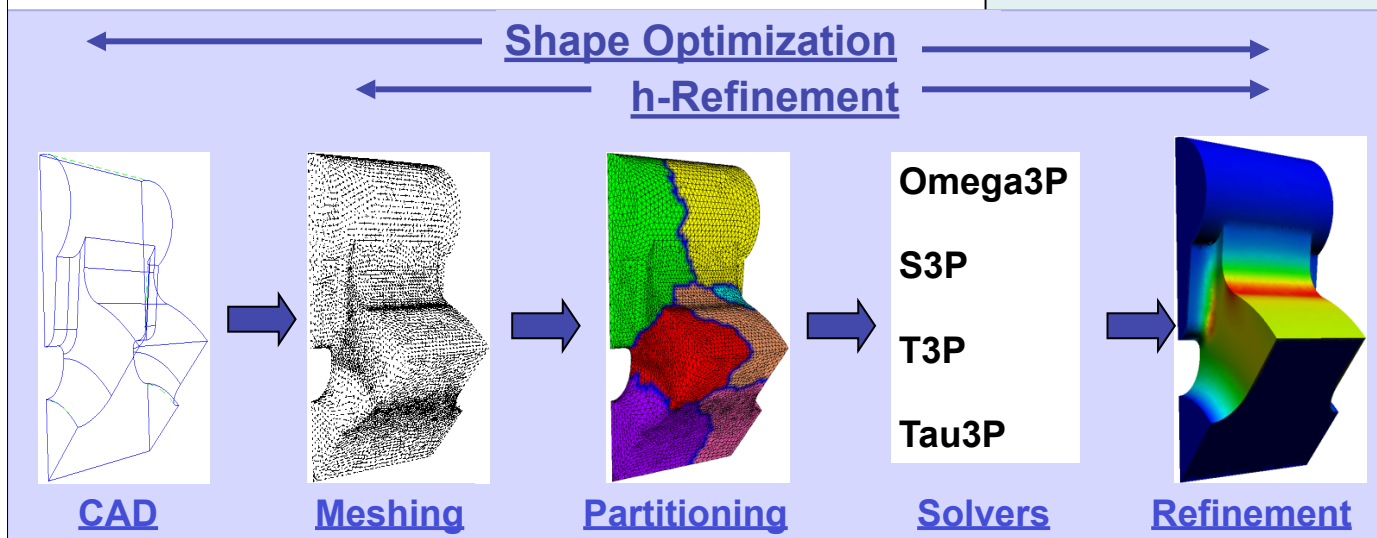
- Focus on interfaces not on data structures or file formats
- Use independent interfaces for distinct data model abstractions to make adoption easier
- Incremental adoption for applications; only service dependent interfaces need to be implemented
- Finer granularity of interoperable functionality reduces the need to mix huge libraries together

ITAPS Data Model

The ITAPS interoperability goal requires abstracting the data model

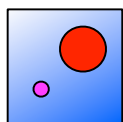
- Information flows from geometrical representation of the domain to the mesh to the solvers and post-processing tools
- Adaptive loops and design optimization requires a loop

- The data model must encompass a broad spectrum of mesh types and usage scenarios
- A set of common interfaces
 - Implementation and data structure neutral
 - Small enough to encourage adoption
 - Flexible enough to support a broad range of functionality

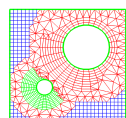


The ITAPS data model abstracts PDE-simulation data hierarchy

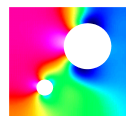
- Core Data Types



- *Geometric Data*: provides a high level description of the boundaries of the computational domain; e.g., CAD, image, or mesh data ([iGeom](#))



- *Mesh Data*: provides the geometric and topological information associated with the discrete representation of the domain ([iMesh](#))



- *Field Data*: provides access to time dependent physics variables associated with application solution. These can be scalars, vectors, tensors, and associated with any mesh entity. ([iField](#))

- Data Relation Managers ([iRel](#))

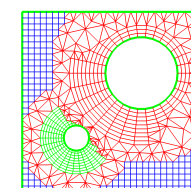
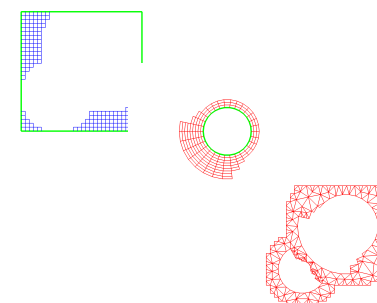
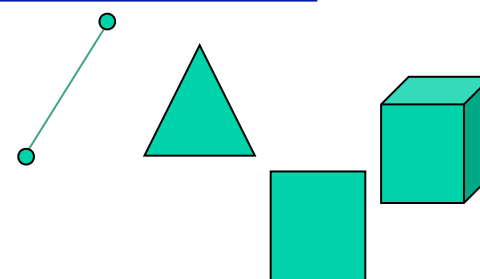
- Provides control of the relationships among the core data types
- Resolves cross references between entities in different groups
- Provides functionality that depends on multiple core data types

The ITAPS data model has four fundamental “types”

- *Entity*: fine-grained entities in interface (e.g., vertex, face, region)
- *Entity Set*: arbitrary collection of entities & other sets
 - Parent/child relations, for embedded graphs between sets
- *Interface Instance*: object on which interface functions are called and through which other data are obtained
- *Tag*: named datum annotated to Entities and Entity Sets

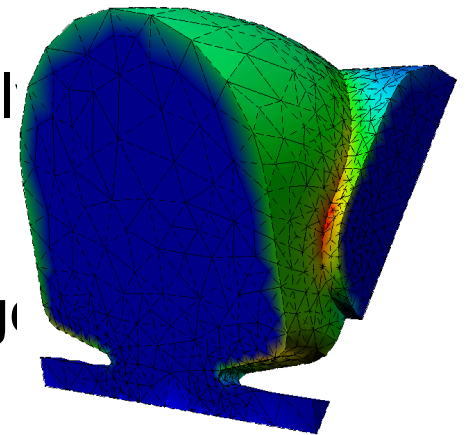
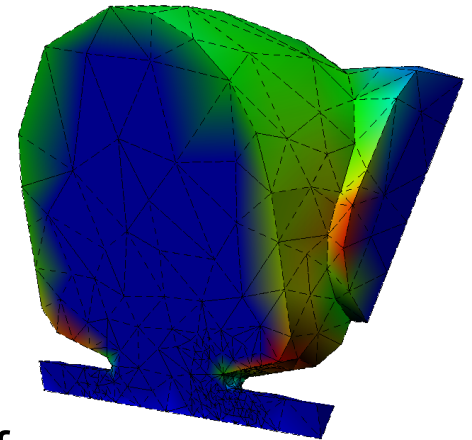
These core data types blend abstract and specific concepts

- Entity Definition
 - Unique type and topology
 - Canonical ordering defines adjacency relationships
- Entity Set Definition
 - Arbitrary grouping of ITAPS entities and other sets
 - There is a single “Root Set”
 - Relationships among entity sets
 - Contained-in
 - Hierarchical
- These objects are accessed using opaque (type-less) “handles”



iMesh(P) provides access to the discrete representation of the domain

- iMesh supports local access to the mesh
- iMeshP complements iMesh with parallel support
- Must support
 - Access to mesh geometry and topology
 - User-defined mesh manipulation and adaptivity
 - Grouping of related mesh entities together (e.g. for boundary conditions)
- Builds on a general data model that is largely suited for unstructured grids
- Implemented using a variety of mesh types, software, and for a number of different usage scenarios

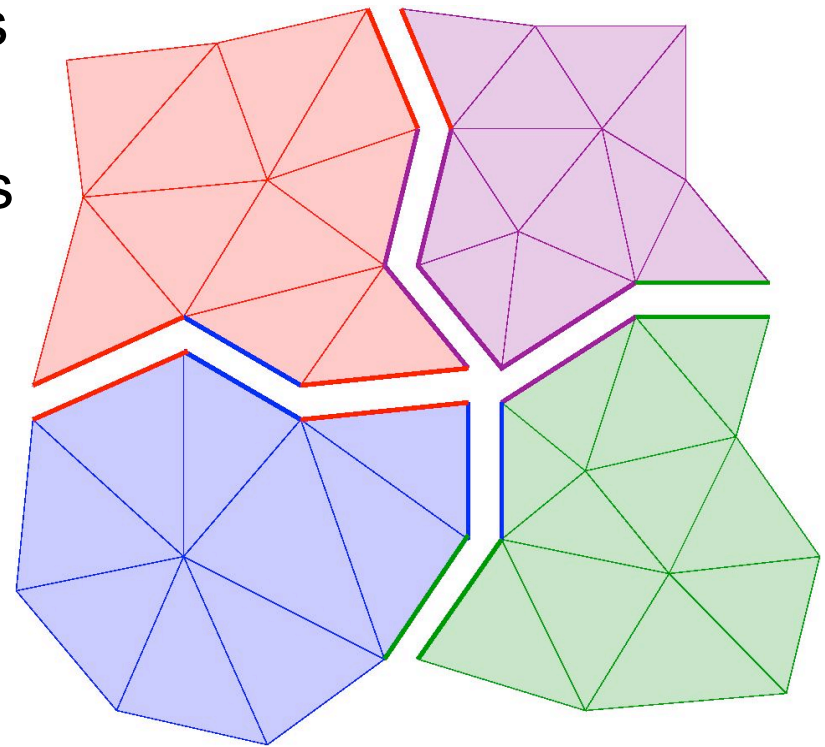


The iMesh interface supports basic and advanced local operations

- Provides basic access to vertex coordinates and adjacency information
 - Mesh loading and saving
 - Global information such as the root set, geometric dimension, number of entities of a given type or topology
 - Access to all entities in a set as single entities, arrays of entities, or entire set
 - Set/remove/access tag data
- Set functionality
 - Boolean operations (union, subtract, intersect)
 - Hierarchical relationships
- Mesh modification
 - Adding / Deleting entities
 - Vertex relocation
 - No validity checks

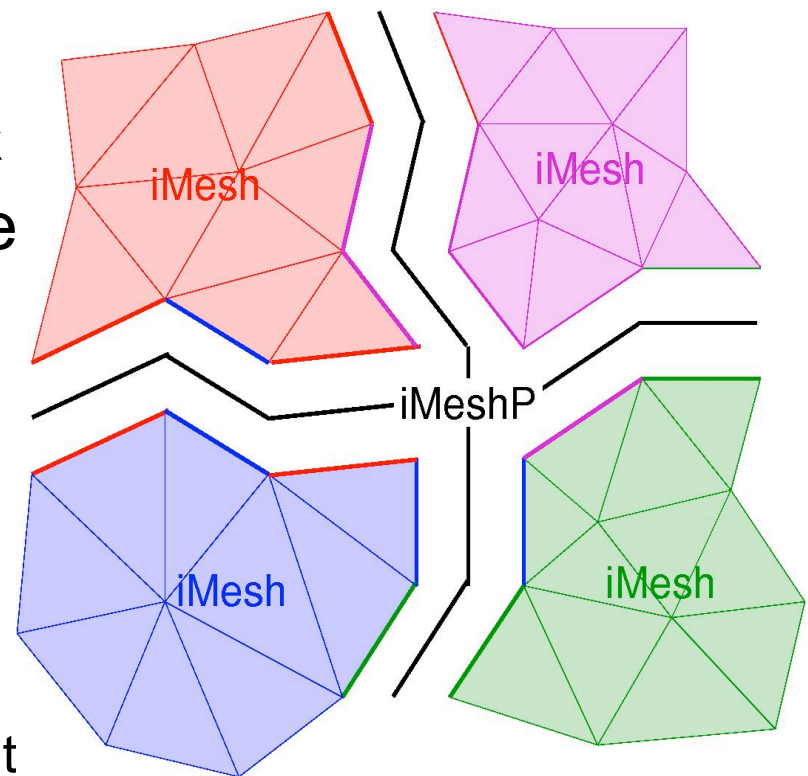
iMeshP extends iMesh to support parallel computations

- Focus on distributed memory
 - For example: use application's MPI communicators
 - But allow use of global address space and shared memory paradigms
- Leverage serial iMesh
 - Works as expected within a process
 - Works as expected for global address space and shared memory programs



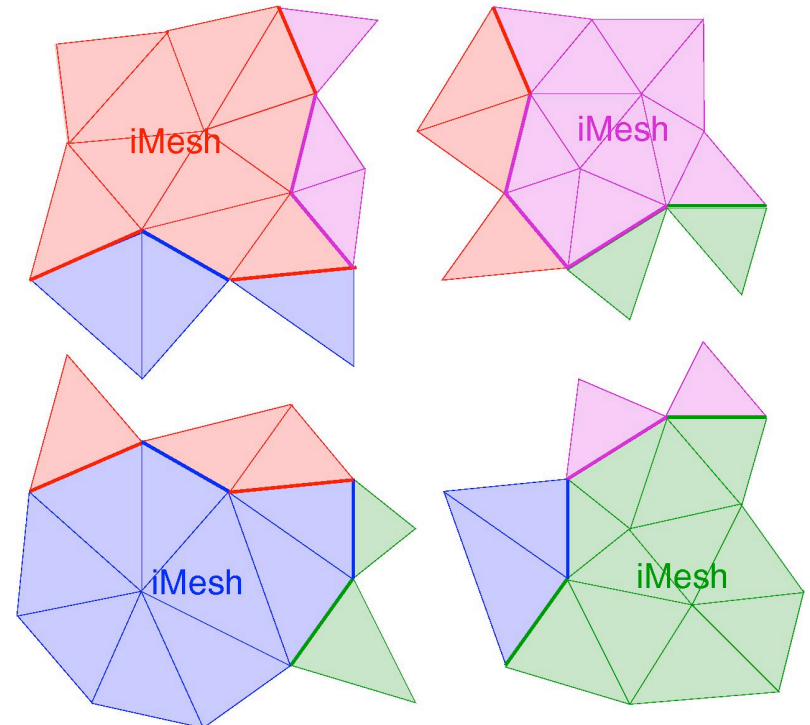
The iMeshP parallel interface defines a partition model

- *Process*: a program executing; MPI process
 - # of processes == MPI_Comm_size
 - Process number == MPI_Comm_rank
- *iMesh instance*: mesh database provided by an implementation
 - One or more instances per process
- *Partition*: describes a parallel mesh
 - Maps entities to subsets called *parts*
 - Maps parts to processes
 - Has a communicator associated with it



The Partition Model

- *Ownership*: right to modify an entity
- *Internal entity*: Owned entity not on an interpart boundary.
 - E.g., all triangles w/ same color as iMesh label for part
- *Part-Boundary entity*: Entity on an interpart boundary
 - E.g., bold edges
 - *Shared* between parts (owner indicated by color; other parts have copies).
- *Ghost entity*: Non-owned, non-part-boundary entity in a part
 - E.g., triangles whose color is different from iMesh label
 - Needed for adjacency and/or solution data
- *Copies*: ghost entities + non-owned part-boundary entities.

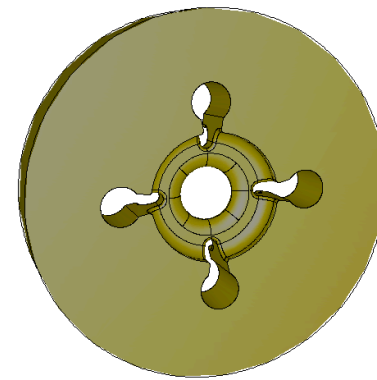


Partition Characteristics

- Maps entities to parts
 - Part assignments computed with respect to a set of entities
 - Computed assignments induce part assignments for adjacent entities
- Maps parts to processes
 - Each process may have one or more parts
 - Each part is wholly contained within a process
- Has a communicator associated with it
 - “Global” operations performed with respect to data in all parts in a partition’s communicator
 - “Local” operations performed with respect to either a part’s or process’ data

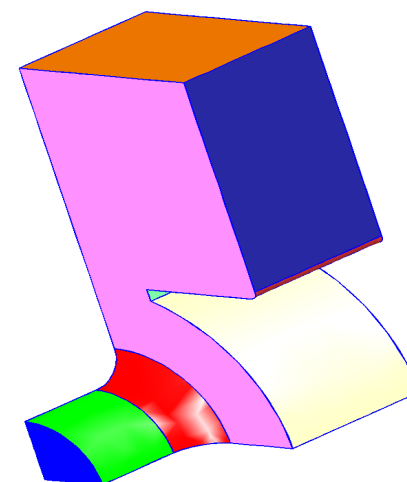
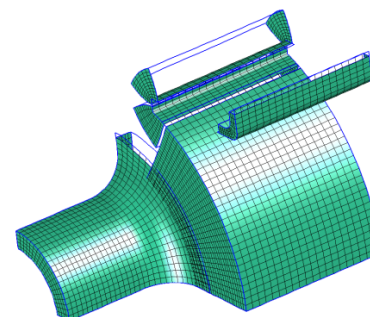
The geometry interface provides access to the computational domain

- Must support
 - automatic mesh generation
 - mesh adaptation
 - tracking domain changes
 - relating information between alternative discretizations
- Builds on boundary representations of geometry
- Used to support various underlying representations
 - Commercial modelers (e.g., Parasolid, ACIS)
 - Modelers that operate from standard files (e.g. IGES, STEP)
 - Models constructed from an input mesh



Basic and advanced functionalities are supported in the geometry interface

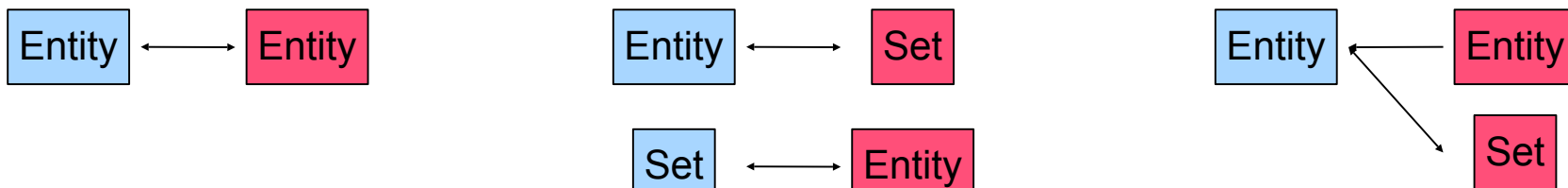
- Model loading and initiation
- Topological queries of entities and adjacencies
- Pointwise geometric shape interrogation
- Parametric coordinate systems
- Model topology modification



iRel Relations interface enables other ITAPS interfaces to work together

- Relates entities between two interfaces without adding dependencies between them. E.g.,
 - Relate entities in iMesh to entities iGeom.
 - Generation of spectral element points on curved geometry.

- Relationships supported:

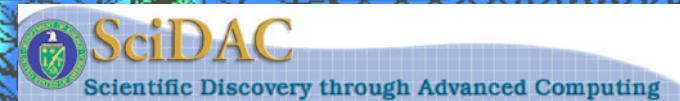


- Implementation available in Lasso.
 - Reference implementation that all services/ implementations can use to manage relationships.

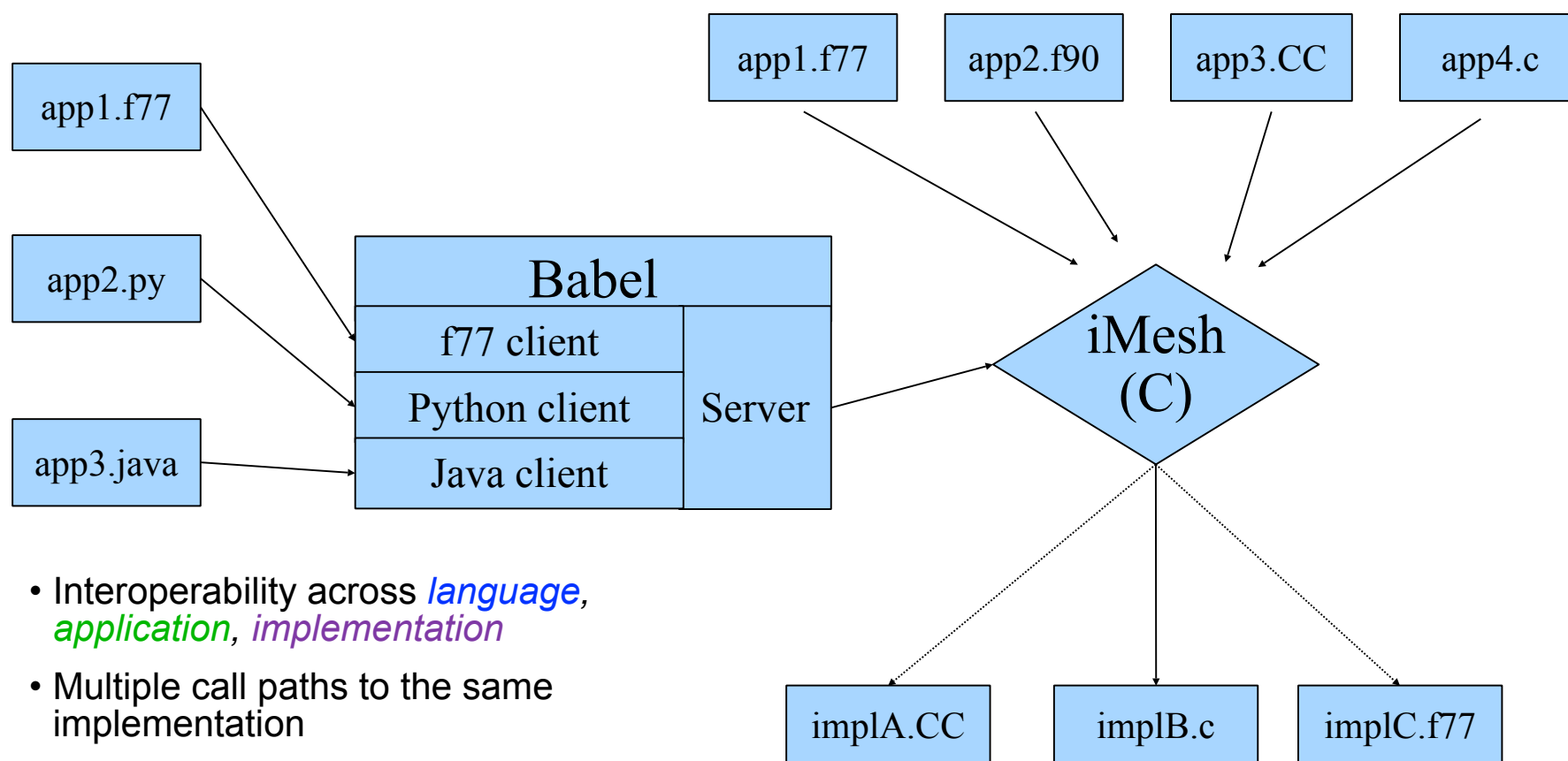
Summary of the ITAPS data model

- The data model abstracts the information flow in PDE simulations: Geometry, Mesh, Fields and their Relations
- Each core abstraction has a separate interface definition: iMesh, iGeom, iField, iRel
- The core building blocks of the data models are entities, entity sets, interfaces and tags
- The parallel data model defines partitions, parts, and entity ownership concepts

Basic ITAPS Interfaces



ITAPS interfaces are designed for interoperability



- Interoperability across *language*, *application*, *implementation*
- Multiple call paths to the same implementation
- Efficiency preserved using direct, C-based interface

Example enumerated types used in the ITAPS interface

- Important enumerated types:
 - EntityType (iBase_VERTEX, EDGE, FACE, REGION)
 - EntityTopology (iMesh_POINT, LINE, TRI, QUAD, ...)
 - StorageOrder (iBase_BLOCKED, INTERLEAVED)
 - TagDataType (iBase_INTEGER, DOUBLE, ENTITY_HANDLE)
 - ErrorType (iBase_SUCCESS, FAILURE, ...)
- Enumerated type & function names both have iBase, iMesh, iGeom, other names prepended

Simple Example: HELLO iMesh

```
#include "iMesh.h"
int main(int argc, char *argv[]){
    char *options = NULL;
    int ierr, dim, num, options_len = 0;
    iMesh_Instance mesh;
    iBase_EntitySetHandle root;

    /* create the Mesh instance */
    1 iMesh_newMesh(options, &mesh, &ierr, options_len);
    iMesh_getRootSet(mesh, &root, &ierr);

    /* load the mesh */
    2 iMesh_load(mesh, root, argv[1], options, &ierr,
               strlen(argv[1]), options_len);

    /* report the number of elements of each dimension */
    3 for (dim = iBase_VERTEX; dim <= iBase_REGION; dim++) {
        iMesh_getNumOfType(mesh, root, dim, &num, &ierr);
        printf("Number of %d-D elements = %d\n", dim, num);
    }
    iMesh_dtor(mesh, &ierr);
    return 1;
}
```

Simple application that

- 1) Instantiates iMesh interface**
- 2) Reads mesh from disk**
- 3) Reports # entities of each dimension**

Note: for brevity, there's no error checking here, but there should be in your code!!!

Simple Example: HELLO iMeshP

```

#include "iMesh.h"
#include "iMeshP.h"
#include <mpi.h>
int main(int argc, char *argv[]) {
    char *options = NULL;
    iMesh_Instance mesh;
    iMeshP_PartitionHandle partition;
    int me, dim, num, ierr, options_len=0;
    iBase_EntitySetHandle root;
    /* create the Mesh instance */
    iMesh_newMesh(options, &mesh, &ierr, options_len);
    iMesh_getRootSet(mesh, &root, &ierr);

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    1 /* Create the partition. */
    iMeshP_createPartitionAll(mesh, MPI_COMM_WORLD, &partition, &ierr);

    2 /* load the mesh */
    iMeshP_loadAll(mesh, partition, root, argv[1], options, &ierr,
                  strlen(argv[1]), options_len);

    /* Report number of Parts in Partition */
    3 iMeshP_getNumLocalParts(mesh, partition, &num, &ierr);
    printf("%d Number of Parts = %d\n", me, num);
  }

```

Parallel Version: HELLO iMeshP

- 1) Instantiates Partition
- 2) Reads mesh into mesh instance and Partition
- 3) Reports # parts in Partition



Interface implementations are well underway

- iMesh 1.0 Interface complete
- iMeshP 0.5 specified and alpha implementations underway
- iGeom and iRel 1.0 interfaces complete
- Implementations
 - iMesh: FMDB, GRUMMP, NWGrid, MOAB
 - iMeshP: FMDB, MOAB
 - iGeom: CGM (Acis, OpenCasCade)
 - iRel: Lasso
- Interfaces have been used to build services and test interoperability
- Analyzing performance when using interface

Four ITAPS mesh components provide iMesh(P) functionality

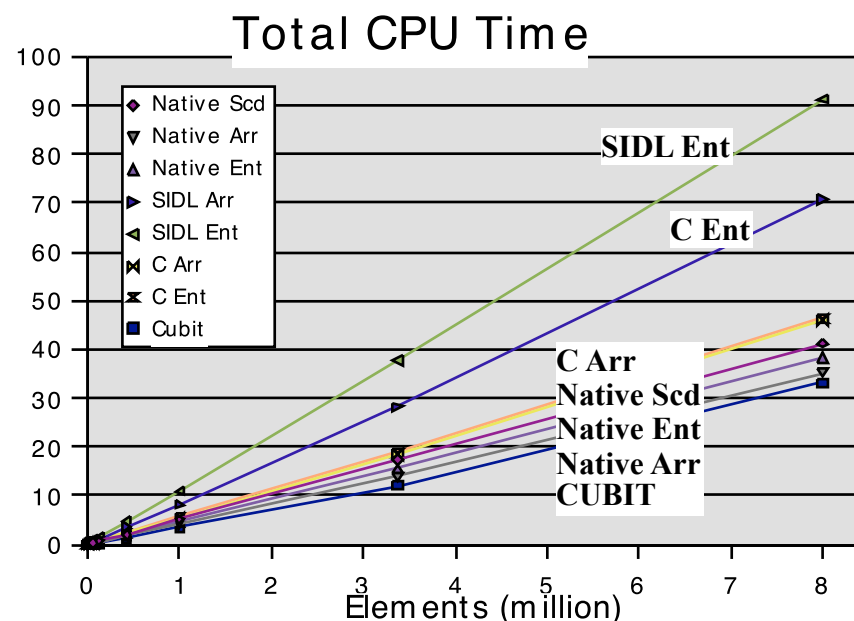
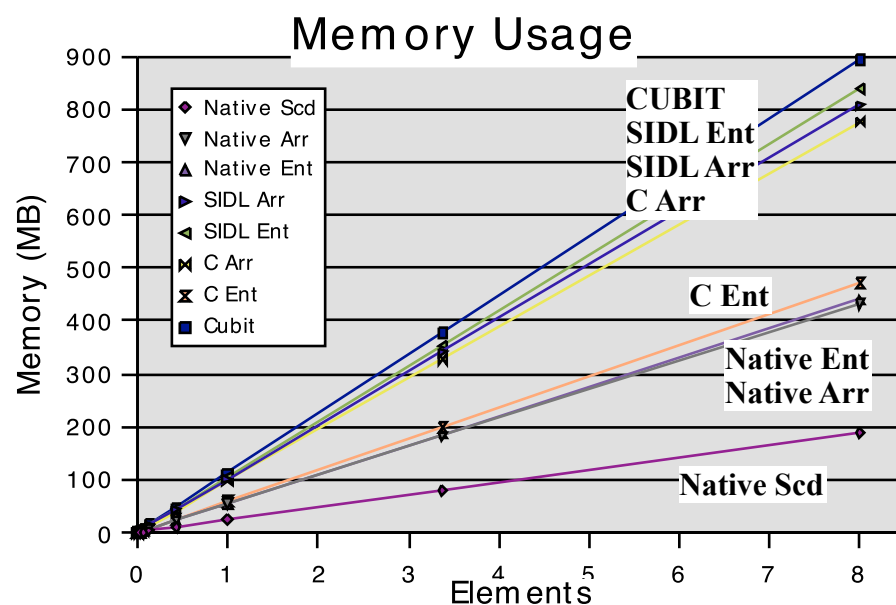
iMesh Implementation	Emphasis	Parallel Capability	Applications
FMDB: Flexible Mesh DataBase	Entity addition/removal for adaptation	Scalable to at least 32K procs, 1B elements; uses iZoltan.	<i>fusion, accelerators</i> , CFD, solid mechanics, multiphase flow
MOAB: Mesh Oriented dAtaBase	Low memory usage first, then CPU time.	Up to 64 procs with flexible mesh loading based on geometric volume, material, partition	<i>nuclear reactors, accelerators</i> , radiation transport, inertial confinement fusion, glacier dynamics
GRUMMP: Generation & Refinement of Mixed-element Meshes in Parallel	Fast adjacency retrieval for mesh generation, improvement and adaptation.	In development.	CFD, biological systems, structural mechanics
NWGrid: NorthWest Grid Generation Code	Simplicial meshes; parallel generation of unstructured, hybrid, adaptive meshes.	Parallelism based on Global Arrays.	CFD, <i>subsurface transport</i>

iGeom implementations provide geometric model data

iGeom Implementation	Capabilities	Applications
CGM: Common Geometry Module	Supports ACIS, Open.Cascade, facet-based engines; Derived from geometry ...	<i>nuclear reactor modeling, accelerator modeling, radiation transport</i>
SGModel: SCOREC Geometric Model	Supports CAD system kernels (ParaSolid, ACIS) and mesh-based geometry.	MeshAdapt service, <i>accelerator modeling, fusion</i> , solid mechanics, multiphase flow.

Performance

- Large applications balance memory and cpu time performance
- Implementations of iMesh vary on speed vs. memory performance
 - Create, v-E, E-v query, square all-hex mesh
 - Entity- vs. Array-based access
- Compare iMesh (C, SIDL), Native (MOAB), Native Scd (MOAB), CUBIT
 - Ent-, Arr-based access
 - All-hexahedral square structured mesh



Performance in building a finite element stiffness matrix

- Set up a simple stiffness matrix for a 2D diffusion equation
- Examine costs of entity access via native data structures, arrays, entity iterators and workset iterators
- Arrays minimize time overhead but require a data copy
- Entity iterators are straightforward to program, minimize memory overhead, but maximize time cost
- Entity array iterators balance time/memory tradeoffs but are the most difficult to program

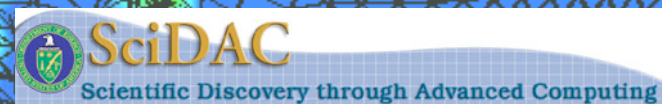
$$\nabla^2 u = f$$

$$u(x=0)=1 \quad u(x=1) = 1$$

$$u_y(x=0, x=1) = 0$$

	Time (ms)	iMesh Overhead
Native	10479	
Array-based	10774	2.8%
Entity Iterator	11642	11.1%
Workset Iterator (1)	11351	8.3%
Workset Iterator (3)	11183	6.7%
Workset Iterator (5)	11119	6.1%
Workset Iterator (10)	11095	5.8%
Workset Iterator (20)	11094	5.8%

An Overview of ITAPS Services



BROOKHAVEN
NATIONAL LABORATORY



Rensselaer



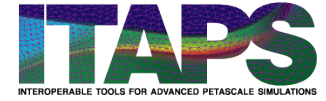
**THE UNIVERSITY OF
BRITISH
COLUMBIA**



Pacific Northwest National Laboratory



Interoperable services speed the development of simulation technologies

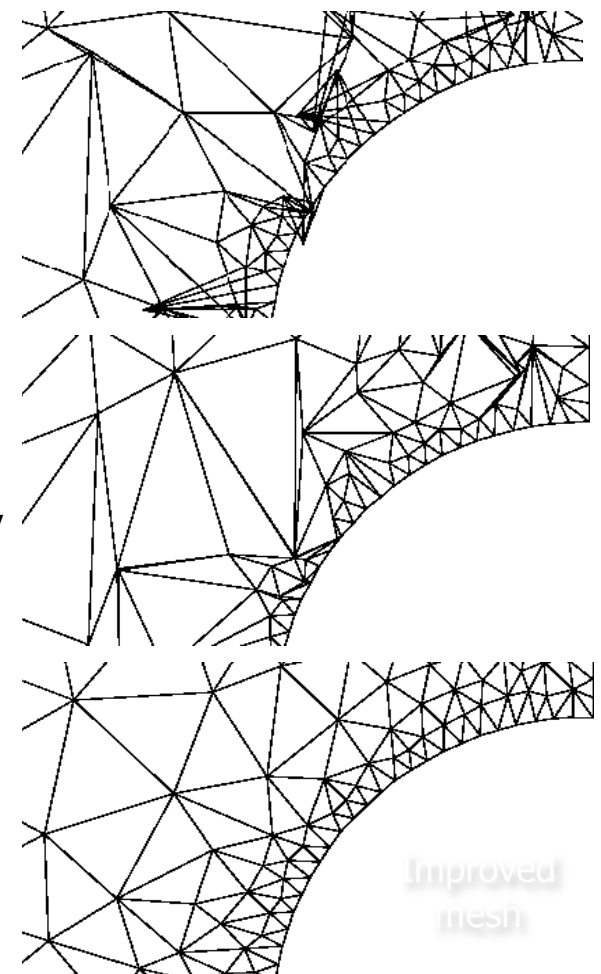


- ITAPS provides stand-alone services as libraries
- Improve applications' ability to leverage advanced tools
 - Mesh quality improvement
 - Mesh adaptation loops
 - Mesh partitioning
 - Front tracking
 - Visualization
 - Mesh I/O

I'll provide a brief overview of each of these tools; more information can be found on www.itaps-scidac.org

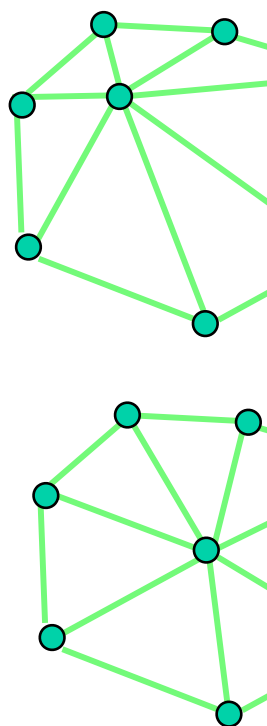
Unstructured mesh quality is a critical factor in solution efficiency and accuracy

- In general, mesh size and quality affects
 - **Solution efficiency** (e.g. Axelsson, 1976; Fried 1972; Axelsson and Barker, 1984)
 - Iterations grow as a function of N
 - Iterations grow as a function of minimum angle
 - **Solution accuracy**
 - Solution from iterative solver is less accurate
 - For isotropic fields, discretization error adversely affected by distorted elements (e.g. Babuska and Aziz, 1976)
- Understanding application solution characteristics is critical
 - stretched elements are more accurate than equilateral elements for boundary layer flow



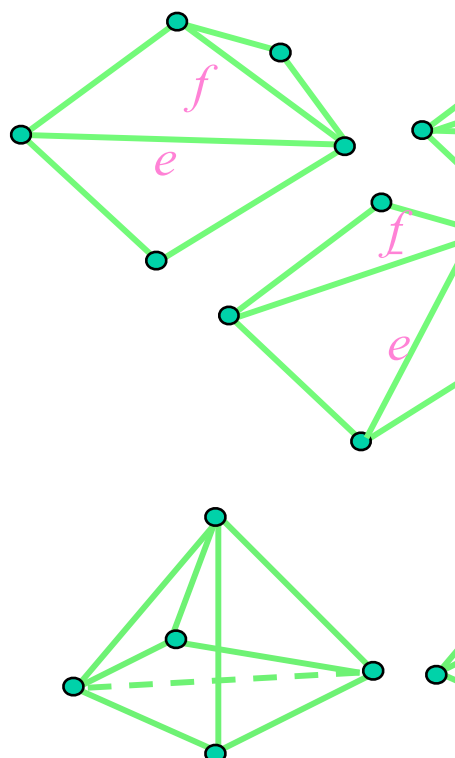
There are three primary techniques for improving the quality of existing meshes

Node Movement



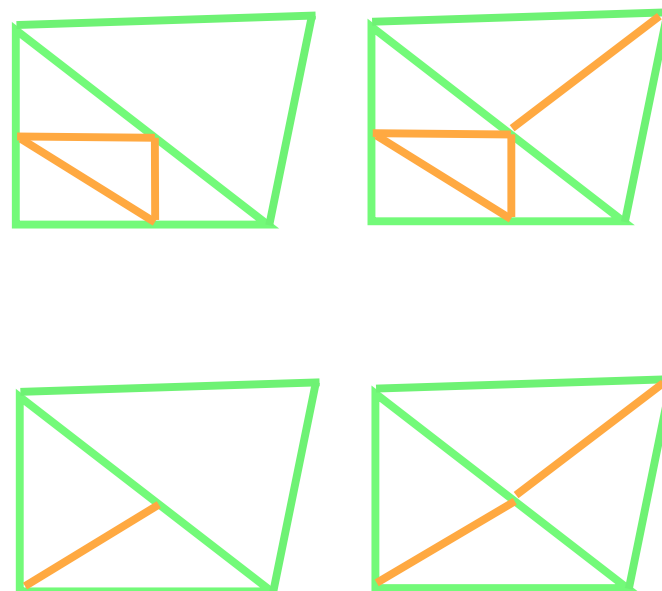
Moving grid point
changing mesh to

Edge or Face Flipping



Modify topology with
grid point location

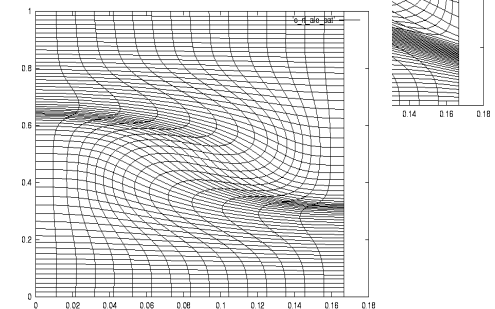
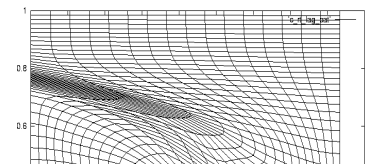
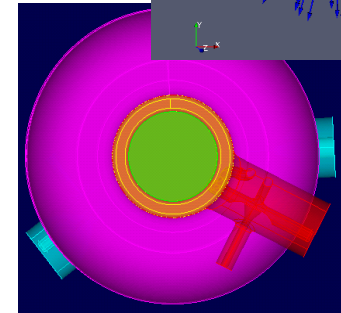
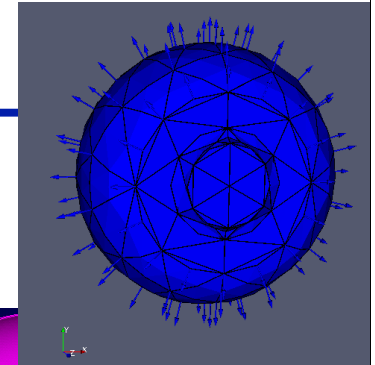
Refinement/Coarsening



Adding or deleting elements
to improve local resolution

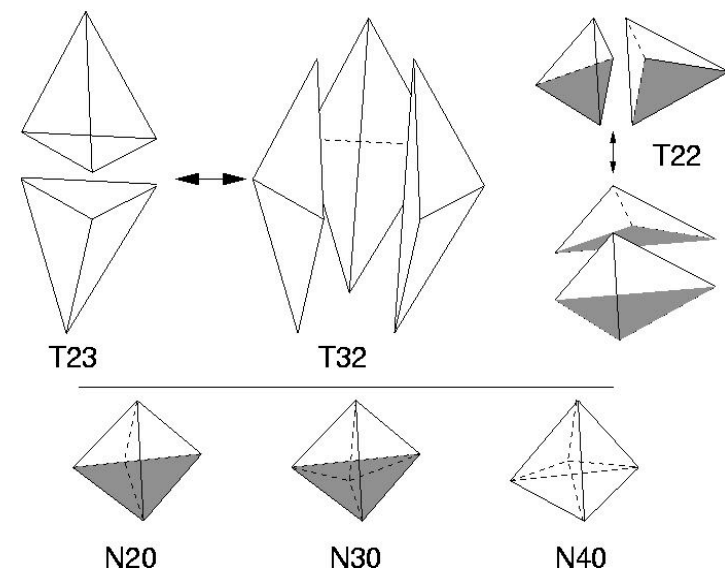
Mesquite provides advanced mesh smoothing capabilities

- Mesquite is a comprehensive, stand-alone library for mesh quality improvement with the following capabilities
 - Shape Quality Improvement
 - Mesh Untangling
 - Alignment with Scalar or Vector Fields
 - R-type adaptivity to solution features or error estimates
- Uses node point repositioning schemes
- Parallel to O(1000) processors
- Tested with FMDB, MOAB, NWGrid, GRUMMP iMesh implementations



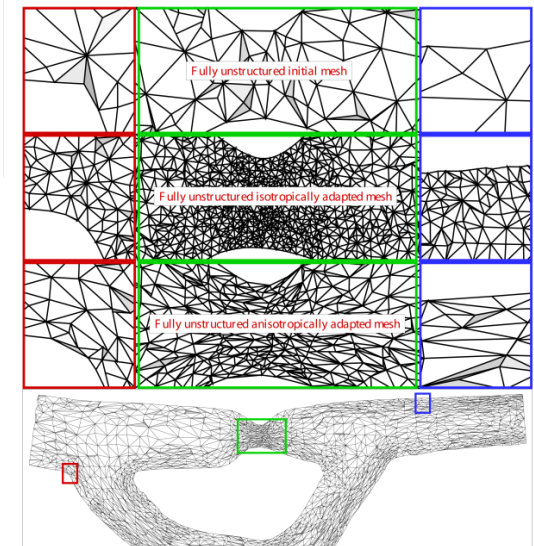
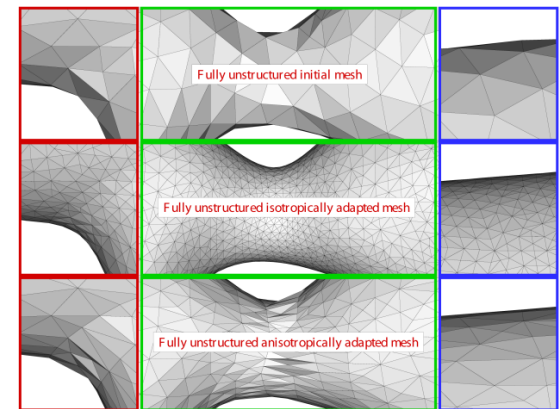
Swapping complements node point movement in improving mesh quality

- Changing topology can eliminate poorly-shaped mesh entities directly
- Service using standard interface handles error-prone aspects of implementation
 - Swapping decisions
 - Topological changes to the mesh
- Swapping service functionality
 - Triangular/tetrahedral edge and face swapping
 - Works on single entities, mesh subsets, or entire mesh
 - Built in and user-defined swapping criteria provide both ease of use and flexibility
- Tested with FMDB, MOAB, and GRUMMP



Mesh adaptation is critical for many applications

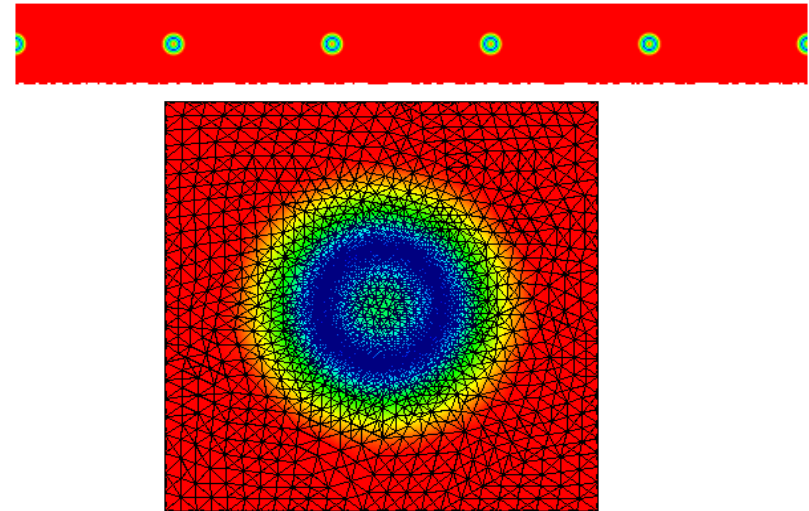
- Determining optimal initial mesh sizes is not possible for complex geometry/ physics
- CFD adaptivity example impact:
 - *Isotropic* : same accuracy as uniform with one order of magnitude fewer elements
 - *Anisotropic* : same accuracy as uniform with two orders of magnitude fewer elements
- ITAPS Mesh Adapt Service
 - Starts with an arbitrary initial mesh with a solution
 - Given a new mesh size field, alters the mesh via local mesh modifications
 - Supports
 - Curved Boundaries
 - Anisotropy
 - Parallel mesh adaptation



Scaling studies of uniform refinement on up to 32K processors

- Weak scaling uniform refinement

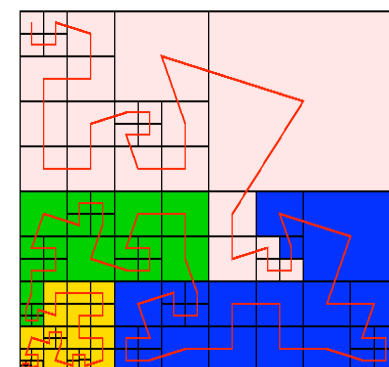
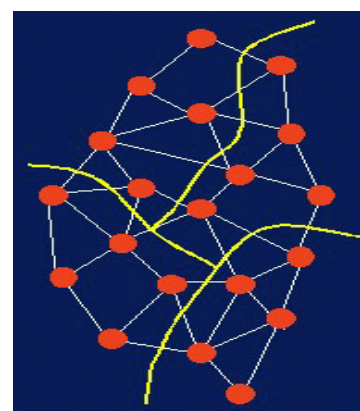
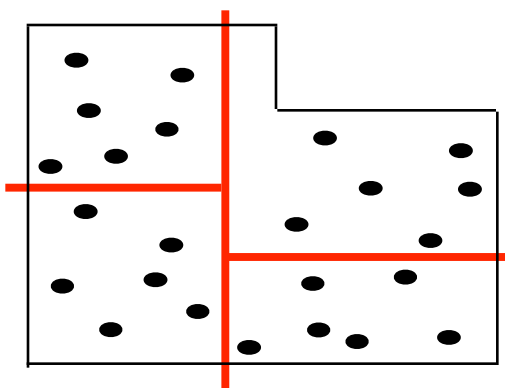
# of Parts	Initial Mesh	Adapted Mesh	Time (s)	Scaling Factor
2048	17M	128M	5.0	1.0
4096	34M	274M	4.8	1.05
8192	65M	520M	5.1	0.97
16384	520M	1.1B	6.1	0.82
32768	274M	2.2B	7.4	0.68



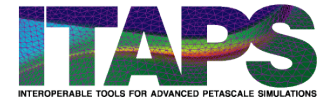
- Mesh adaptation characterized by small, but variable, work per operation - “perfect scaling” too costly
 - Can run on the large numbers of parts used in the analysis
 - Will still be a small % of total solution time - the mesh adapt example given is 0.04% of the estimated solve time
- Improvements to message passing can improve scaling

Zoltan partitioners can now access mesh data through ITAPS interfaces

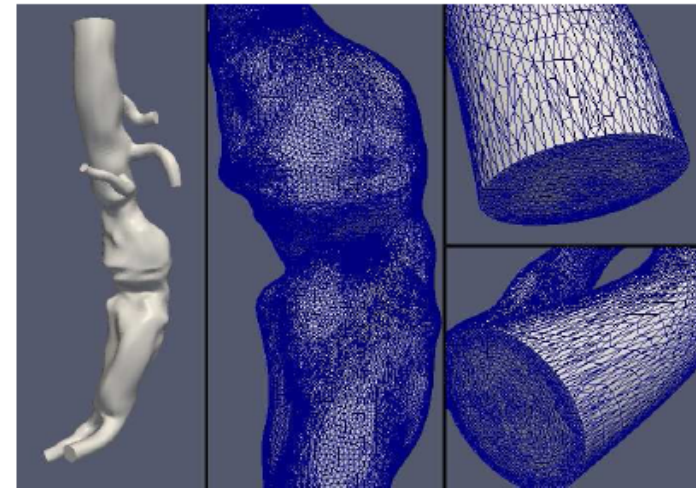
- Partitioning Methods
 - Geometric (RCB, Space filling curves)
 - Connectivity-based (ParMetis, Scotch, Hypergraph)
- iMesh and iMeshP versions available
- Tested and adopted by FMDB, GRUMMP, MOAB, MeshAdapt, accelerator and fusion scientists



MeshAdapt and iZoltan used to prepare strong scaling study on 128K processors



- PHASTA CFD solver
 - Implicit time integration - iterative system solution at each time step
 - Employs the partitioned mesh for system formulation and solution
- PHASTA's work characterized as:
 - Organized and regular communication between parts that “touch” each other
 - A specific number of ALL-REDUCE communications also required
- ITAPS Services used
 - FMDB for the mesh database
 - MeshAdapt for refinement up to 32K
 - iZoltan to partition the mesh to 128K
- Strong scaling highlights need for advanced partitioning algorithms

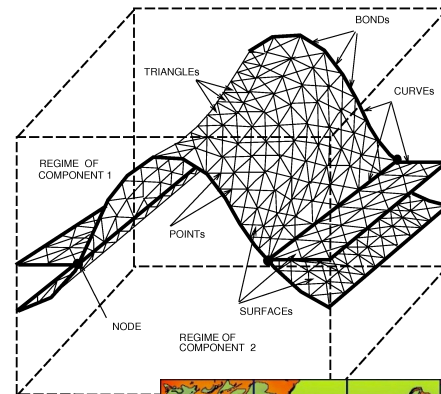


1 billion element anisotropic mesh on Intrepid Blue Gene/P

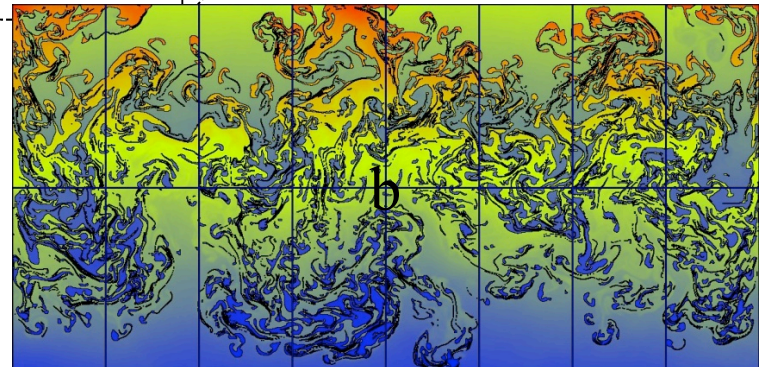
#of cores	Rgn imb	Vtx imb	Time (s)	Scaling
16k	2.03%	7.13%	222.03	1
32k	1.72%	8.11%	112.43	0.987
64k	1.6%	11.18%	57.09	0.972
128k	5.49%	17.85%	31.35	0.885

Interface tracking is available through the FronTier library

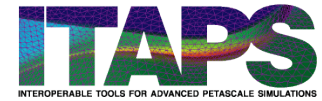
- Intended uses include
 - Computational domains sharply different quantitatively or qualitatively, boundary dynamic
 - Tracking the dynamic motion of distinct bodies, or interfaces between distinct physical regions
- Coupled with hyperbolic, parabolic, and elliptic PDE solvers
- Parallel to 16K processors
- *DOE Applications*: fluid-fluid, fluid-structure, crystal growth, phase transition, elastic-plastic, and other moving interfaces
- Available and tested with the iMesh interfaces through FMDB, MOAB, and GRUMMP



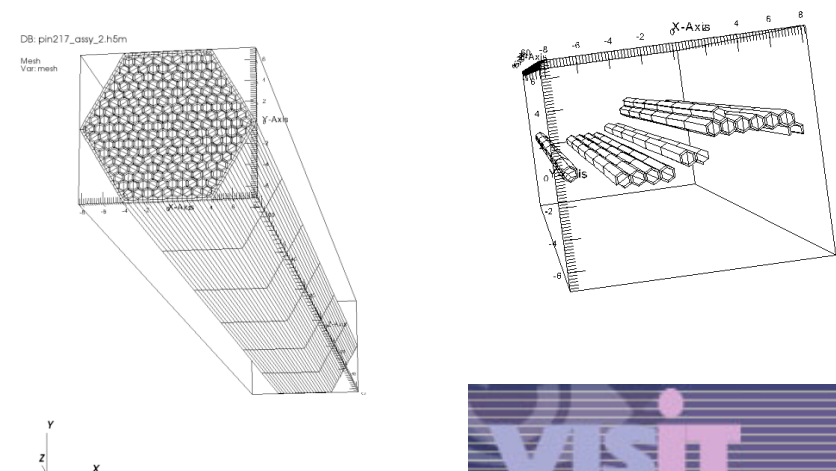
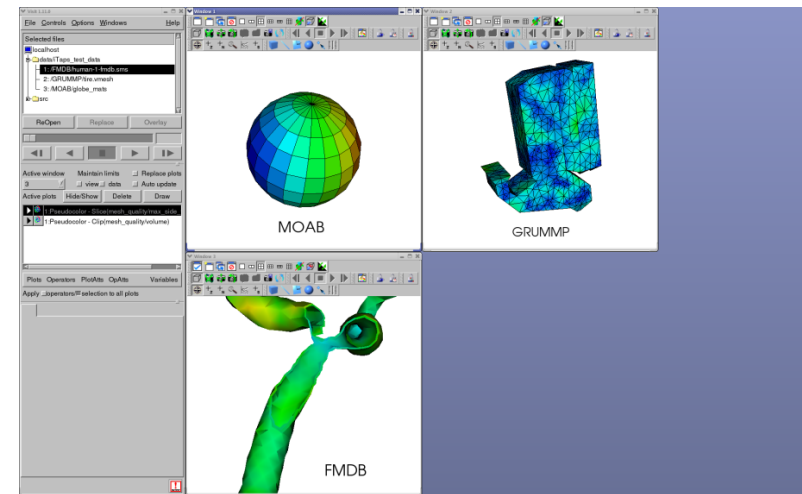
FronTier meshed data structure



ITAPS has been integrated with VisIt as a database plug-in for visualization

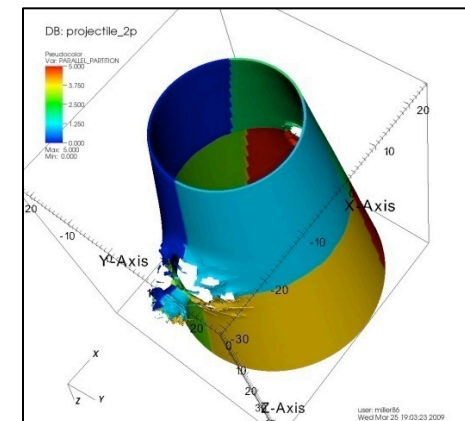
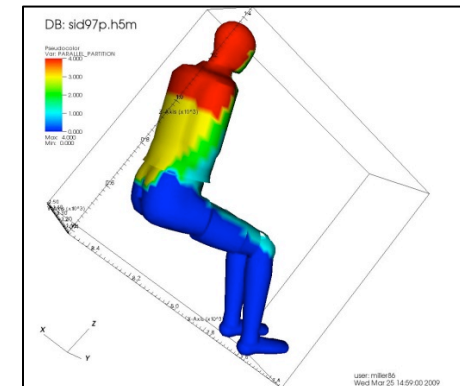


- A single plug-in supports multiple ITAPS implementations
- Supports all entity types, set and tag data through iMesh
- Prototype parallel service demonstrated with iMeshP and FMDB
- Future integration will use VisIt's in-situ 'simulation' interface for run-time vis



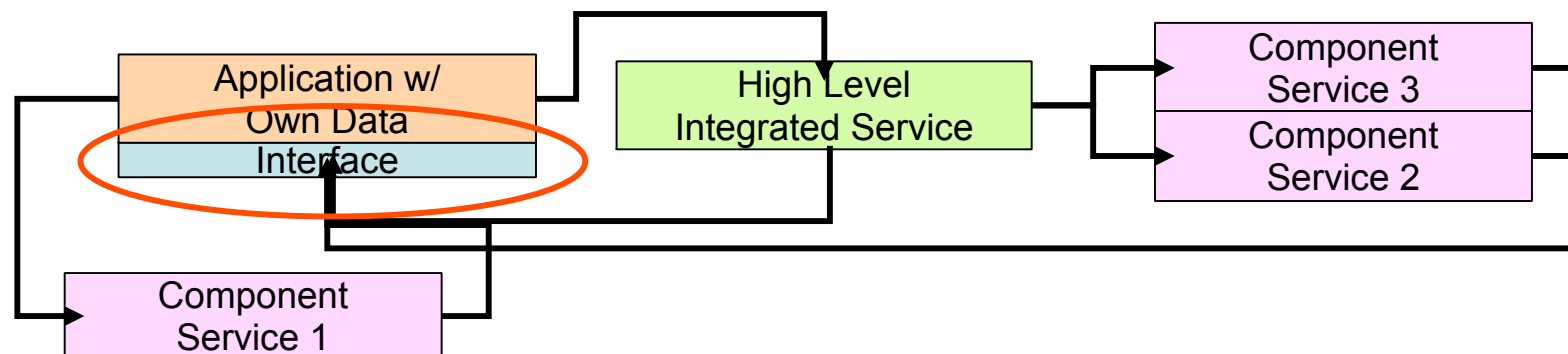
Interoperable Mesh I/O is a key new service to broaden interoperability

- Provides access to a wide array of scientific data through VisIt readers
 - 100+ different file formats
 - Instantiated into any iMesh implementation
- Supports all mesh topology and geometries, and fields stored as dense tags
- Demonstrated ability to instantiate data from previously unsupported file type and partition it via iZoltan through iMesh



Implementing ITAPS interfaces allows use of services on your data structures


- Need to implement some ITAPS interface functions using your data structures – BUT NOT ALL
- Most ITAPS functions reflect things you already do with your databases; most implementation tasks are a thin wrapper
- Compliance testing tools ensure correctness
- Can use a reference implementation at the cost of a data copy to experiment with services



Interface functions needed by the various services

	Coordinates	Adjacency	Other queries	Iterators	Modification	Basic Sets	Tags	Parallel	iGeom/iRel	Total
Mesquite	1	1	4	4	1	2	13		6	32
Swapping	1	1	4	4	2					12
Mesh Adapt	1	2	5	3	3		7	13		34
FronTier Lite	3	1	5	3	3		3			18
Zoltan	1	2	5					14		22
VisIt	1	1	9	3		1	16			31

ITAPS Software

 **SciDAC**
Scientific Discovery through Advanced Computing



Rensselaer



THE
UNIVERSITY OF
BRITISH
COLUMBIA

BROOKHAVEN
NATIONAL LABORATORY

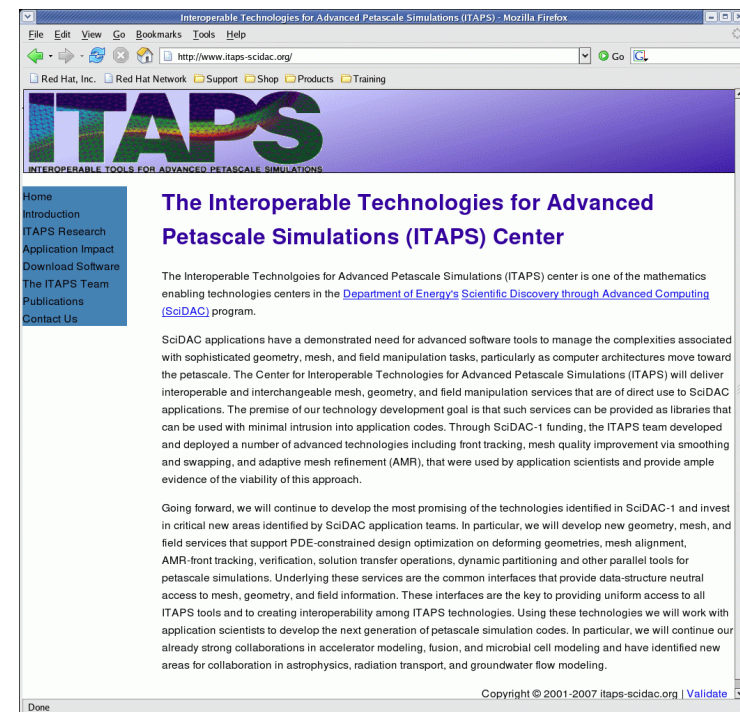
 OAK RIDGE NATIONAL LABORATORY

 Pacific Northwest National Laboratory

ITAPS Web Pages

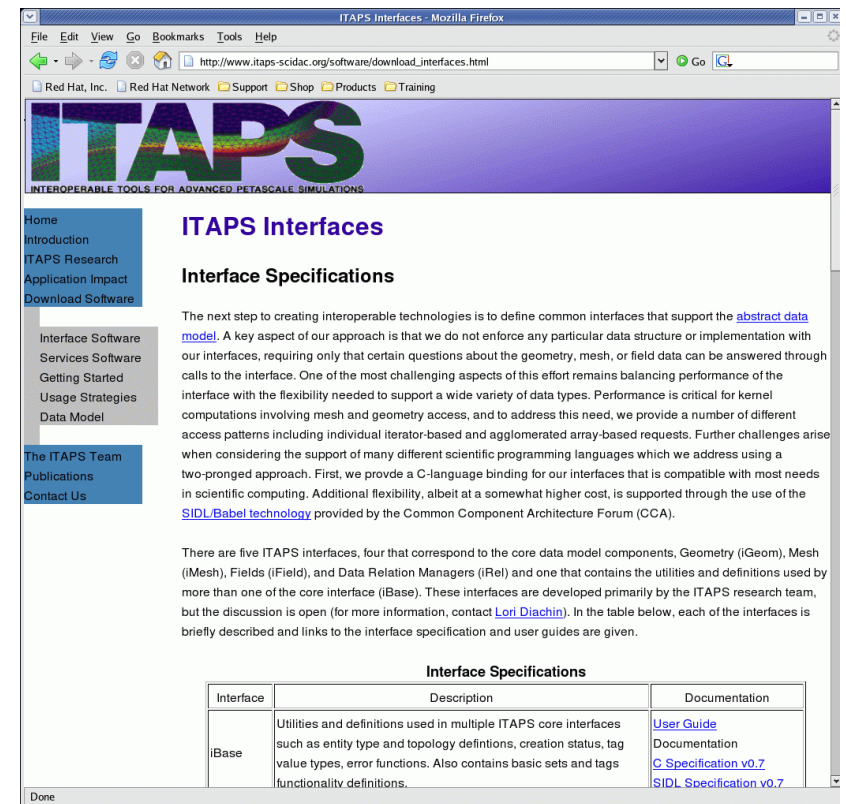
<http://www.itaps-scidac.org>

- Provides help getting started
- Usage strategies
- Data model description
- Access to interface specifications, documentation, implementations
- Access to compatible services software



Interface Software Access

- Links to the interface user guides and man pages where available
- Links to implementations for iMesh, iGeom, iRel
 - Version 1.0 compatible software
 - Links to the home pages for more information
- Simple examples, compliance testing tools and build skeletons



ITAPS Interfaces

Interface Specifications

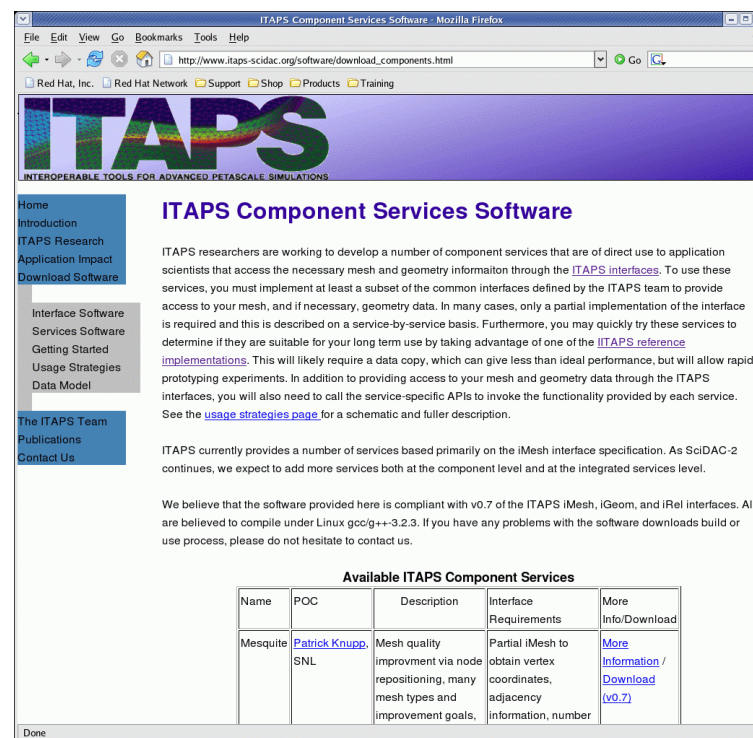
The next step to creating interoperable technologies is to define common interfaces that support the [abstract data model](#). A key aspect of our approach is that we do not enforce any particular data structure or implementation with our interfaces, requiring only that certain questions about the geometry, mesh, or field data can be answered through calls to the interface. One of the most challenging aspects of this effort remains balancing performance of the interface with the flexibility needed to support a wide variety of data types. Performance is critical for kernel computations involving mesh and geometry access, and to address this need, we provide a number of different access patterns including individual iterator-based and agglomerated array-based requests. Further challenges arise when considering the support of many different scientific programming languages which we address using a two-pronged approach. First, we provide a C-language binding for our interfaces that is compatible with most needs in scientific computing. Additional flexibility, albeit at a somewhat higher cost, is supported through the use of the [SIDL/Babel technology](#) provided by the Common Component Architecture Forum (CCA).

There are five ITAPS interfaces, four that correspond to the core data model components, Geometry (iGeom), Mesh (iMesh), Fields (iField), and Data Relation Managers (iRel) and one that contains the utilities and definitions used by more than one of the core interface (iBase). These interfaces are developed primarily by the ITAPS research team, but the discussion is open (for more information, contact [Lori Diachin](#)). In the table below, each of the interfaces is briefly described and links to the interface specification and user guides are given.

Interface Specifications		
Interface	Description	Documentation
iBase	Utilities and definitions used in multiple ITAPS core interfaces such as entity type and topology definitions, creation status, tag value types, error functions. Also contains basic sets and tags functionality definitions.	User Guide Documentation C Specification v0.7 SIDL Specification v0.7

Services Software Access

- Links to the services built on the ITAPS interfaces
- Currently available
 - Mesquite
 - Zoltan
 - Swapping
 - Frontier
 - VisIt Plug In
- Links to home pages for more information
- Instructions for build and links to supporting software



The screenshot shows the ITAPS Component Services Software web page. The page has a navigation menu on the left with links: Home, Introduction, ITAPS Research, Application Impact, Download Software, Interface Software, Services Software, Getting Started, Usage Strategies, Data Model, The ITAPS Team, Publications, and Contact Us. The main content area is titled "ITAPS Component Services Software" and contains text explaining the purpose of the services and the current state of development. It mentions that ITAPS researchers are working on component services for application scientists, requiring mesh and geometry data. It also notes that the software is compliant with v0.7 of the ITAPS iMesh, iGeom, and iRel interfaces and is believed to compile under Linux gcc/g++-3.2.3.

Available ITAPS Component Services

Name	POC	Description	Interface Requirements	More Info/Download
Mesquite	Patrick Knupp , SNL	Mesh quality improvement via node repositioning, many mesh types and improvement goals.	Partial iMesh to obtain vertex coordinates, adjacency information, number	More Information / Download (v0.7)

ITAPS Software: Best Practices

- Use C-based interface where possible, for efficiency
- Pre-allocate memory in application or re-use memory allocated by implementation
 - E.g. getting vertices adjacent to element – can use static array, or application-native storage
- Take advantage of implementation-provided capabilities and iMesh compliant services
 - Partitioning, IO, parallel communication, (parallel) file readers
- Try different implementations: they are tuned for different application uses – Experiment!
- Implement iMesh on top of your data structure
 - Take advantage of tools that work on iMesh API
- Let us help you
 - Not all best practices are easily described or self-evident

Conclusions

What you learned today

- A component-based approach to mesh services and tools is both flexible and effective
- The ITAPS interfaces provide an avenue to leverage many existing technologies and a path to incremental adoption

None of this would be possible without a strong team of contributors

I thank all those who have contributed to the ITAPS interface definition effort and software!

- **ANL: Tim Tautges**
- **LLNL: Lori Diachin, Mark Miller, Kyle Chand, Martin Isenburg**
- **PNNL: Harold Trease**
- **RPI: Mark Shephard, Ken Jansen, Eunyoung Seol, Xiaojnan Luo, Ting Xie, Onkar Sahni**
- **SNL: Vitus Leung, Karen Devine**
- **SUNY SB: Xiaolin Li, Brian Fix, Ryan Kaufman**
- **UBC: Carl Ollivier-Gooch**
- **U Wisconsin: Jason Kraftcheck, Jane Hu**

Contact Information

- ITAPS Web Page: <http://www.itaps-scidac.org>
- ITAPS Software Page: <http://www.itaps-scidac.org/software>
- Email: itaps-mgmt@llnl.gov
- Tutorial Presenter:



Lori Diachin, LLNL
diachin2@llnl.gov

Auspices and disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.